EZHPC: Easy Access to High Performance Computing

Patti Duett, Wes Monceaux, and Keith Rappold
USACE Engineer Research and Development Center (ERDC), Vicksburg, MS
{Patti.S.Duett, Weston.P.Monceaux, Keith.N.Rappold}@erdc.usace.army.mil

Abstract

EZHPC is a Simple Object Access Protocol (SOAP)-based Web service that represents an Application Programming Interface (API) for securely accessing and manipulating high performance computing (HPC) resources. This service was created to make it "easy" for users who are unfamiliar with the HPC environment but would benefit from the utilization of these resources. It not only promotes HPC resources but also allows for the development of future applications through the reuse of this architecture.

The API is exposed to developers as a SOAP-based Web service. Numerous API method calls are available, some of which include, but are not limited to, the ability to authenticate users against the HPC Kerberos system, move files to and from the HPC systems, submit and monitor batch jobs, access the Data Management System (DMS), etc. The Web service paradigm was chosen so that any type of user front end could be constructed to interact with the HPC resources. This includes stand alone user applications, Personal Digital Assistant (PDA)-based applications, Web applications, or inclusion in existing applications (ArcInfo, Watershed Modeling System (WMS), Groundwater Modeling System (GMS), Surface-Water Modeling System (SMS), etc. A fully functional client application has been developed to demonstrate the usage of the API.

The authors will present a detailed inspection of the EZHPC architecture. The structure, usage, and security implications are described for each component of the system. In addition, the HPC-issued Kerberos user utilities, the Web server components, and the database component will be discussed. The client reference implementation will also be covered. The presentation will conclude with a summary of future enhancements and how these enhancements will expand the functionality of EZHPC, increasing its value to the user community.

1. Introduction

EZHPC attempts to make accessing HPC resources easier by providing an Application Programming Interface (API) for securely accessing and manipulating HPC resources. The API is represented as a SOAP-based Web service that provides many method calls to developers for utilizing HPC resources. They include the ability to authenticate users against the HPC Kerberos system, move files to and from the HPC systems, submit and monitor batch jobs, access the Data Management System (DMS), and others.

By creating an API, the user interface is completely separated from the functionality of the architecture. This allows access to HPC resources to be included in new, and added to existing, applications on various platforms and form factors. Developers could use the API with stand alone user applications, Personal Digital Assistant (PDA)-based applications, Web applications, or inclusion in existing applications (ArcInfo, Watershed Modeling System, Groundwater Modeling System, Surface-Water Modeling System, etc.). A functional user interface has been constructed to demonstrate the usage of this API.

This paper presents a detailed inspection of the EZHPC architecture. The structure, usage, and security implications are described for each component of the system. In addition, the HPC-issued Kerberos user utilities, the Web server components, and the database component will be discussed. The client reference implementation will also be covered.

2. Kerberos Utilities

2.1. Description.

The Kerberos utilities used are the Linux binary versions of the HPC-issued Kerberos and SSH utilities. No modifications have been made to the tools. They are used "as is." The primary benefit of using the standard

Kerberos tools is that if the tools are updated, few, if any, coding changes are necessary to utilize new versions of the tools.

2.2. Structure.

The Kerberos tools are installed on the host Linux system and placed in the environment PATH of the Web server user. The Kerberized versions of the tools must come first in the user's PATH. This ensures that the proper versions of the tools are invoked. If not properly set incorrect versions of SSH and other tools will be used and will generally fail to function as expected.

2.3. Usage.

The Web server is used to invoke a number of wrapper scripts, which in turn call the various Kerberos tools. These are Bourne shell scripts that take a number of parameters necessary to set up the proper environment variables to point to the current user's Kerberos ticket cache. The command to the HPC system can then be executed on behalf of the user. An Expect script is used for calling the kinit command.

2.4. Security.

Aside from ease of upgrading tool chains, the choice of using the standard Kerberos tools means that there are no new security issues introduced by using nonstandard Kerberos utilities.

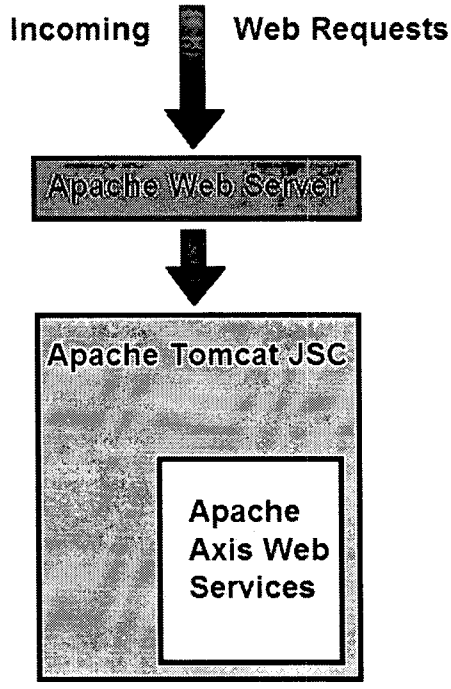
3. Web Server Components

3.1. Description.

This system is designed to be Web-based. All communication between a client and the HPC systems happens via the Web server components. Three software components make up the Web server components; these are the Apache Web server, Apache Tomcat Java Servlet Container, and the Apache Axis Web Service library.

3.2. Structure.

The Web server components are organized as seen in the following diagram.



The Apache Web Server (Apache) receives all incoming Web requests. The Apache Tomcat Java Servlet Container (Tomcat) runs as a separate process that is not accessible directly by the network. Apache forwards all requests to Tomcat. For those Web requests destined for the API Web service, Tomcat invokes the proper code within the Apache Axis Web Services (Axis) Web application. Axis runs as code within the Tomcat process.

The Web requests destined for the API Web service are SOAP requests, XML messages transported over HTTP. A SOAP request first arrives at Apache, which is then forwarded to Tomcat, which processes it into appropriate calls to the Axis code, which in turn invokes the API Web service code. Depending on the contents of the SOAP request, different actions are taken by the API Web service. Most actions involve invoking the Kerberos wrapper scripts to execute a command on an HPC system. The results of the SOAP request are returned back up the chain to the user as a SOAP response.

This structure is a Web-based remote-procedure-call (RPC) implementation. This is the basis for how all commands are issued to the API.

3.3. Usage.

The Apache, Tomcat, and Axis components will remain dormant until something invokes methods from the API Web service. Methods are invoked by SOAP-aware applications. SOAP is a programming language-neutral-protocol. Several language libraries exist for it, including Microsoft .NET (C#, VB, etc.), Java, Python,

PHP, Perl, and numerous others. Axis is able to generate Web Service Description Language (WSDL) descriptions of the API Web service automatically. WSDL is an XML-based description of the available methods and parameters for a Web service. Using a WSDL file, most SOAP implementations can generate a library of stub and header files (or equivalent) for invoking the Web service. Basically, this means that all that needs to be provided to other developers is the WSDL description of the API Web service for them to use it. There is no need to distribute code libraries.

3.4. Security.

Tomcat is actually able to run as a standalone Web server. Apache is layered on top because it provides better control over Secure Socket Layer (SSL) parameters. Apache is used to enforce AES or Triple DES encrypted SSL connections to the Web server. Clients connect to Apache using AES or 3DES SSL, and the requests are then forwarded (locally, not over the network) to Tomcat unencrypted.

The initial connection to the Web server is a presentation of authentication tokens that are used to try to authenticate the user. If kinit fails, an HTTP level error is reported to the client. If kinit succeeds, subsequent requests are honored and passed along to the API Web service. The generated granting ticket is passed back to the client to be used in future requests. The ticket is not stored on the Web server system. If a client submits a ticket as part of the HTTP authentication, it is used to process the request. All calls to the API Web service are logged.

4. Database

4.1. Description.

The database used is MySQL. It runs on the Linux system along with the Web server components. It is not accessible over the network. It is used to hold configuration information for all "configured" HPC systems. It also holds nonsensitive user information, such as job run statistics, application preferences, etc.

4.2. Structure.

The MySQL database is used to hold all configuration information for the API Web service. The API Web service provides a uniform method interface across multiple HPC systems by creating mappings from the general desired functionality of the API to the specific implementation of that functionality on each HPC system. For instance, an API call to retrieve a directory listing makes use of the 'ls' command on most HPC systems. However, an HPC system existing that uses some other command to retrieve a directory listing, such as 'dir' is conceivable. The database is used to maintain the function-to-command mappings for each HPC system. Each HPC system must have similar configuration information placed into the database before the API Web service will know how to properly interact with it.

The other type of information stored in the database is the user-centric data. The database is used to hold statistics on all job runs submitted through the API Web service. The following data are stored in the database:

- User Principle
- HPC System configuration needed for EZHPC operation (e.g., location of utility binaries such as ls, unzip, gzip, and bsub)
- User job information (e.g., job submission time, job name, and job status)
- User-defined application scripts (e.g., bsub scripts for running a specific model)

4.3. Usage.

The database is configured to communicate only with the Web service. When an API Web service method is called, the database is consulted to determine the proper commands to execute based on the selected host.

4.4. Security.

The database is protected from direct client access. Network connections are not permitted to the database. Database connections are limited to the local system. Arbitrary database queries are not permitted via the API Web service.

5. Client Reference Implementation

5.1. Description.

The EZHPC Client is a reference implementation of a client that makes use of the API Web service. In addition to being a proof of concept, the client is designed to be a solid, easy-to-use front end for widespread use.

5.2. Structure.

The reference implementation client is a Windows-based application. It is designed for ease of use for common user-oriented actions on the HPC systems. The client performs all of its interaction with the HPC systems via the API Web service. The client is, therefore,

considered a reasonably thin client. The bulk of the client processing involves accessing local PC resources (files, etc.). Otherwise, the API Web service does most of the work. The user, of course, will need network access, and the ability to connect to the API Web service. The client needs only to have network access to the API Web service system and not any of the HPC systems, as all HPC system interaction is performed by the API Web service on behalf of the user.

5.3. Usage.

Using the client, users are able to log in using Kerberos, move files to and from HPC systems, submit and monitor jobs, access the Data Management System (DMS), create and save commonly used batch scripts, and view job history statistics.

5.4. Security.

The EZHPC API service is designed to be secure regardless of whether a secure or an insecure client connects to it. In securing the API Web service, future clients of the API service will be implicitly secure in accessing HPC resources. This should more easily enable other clients to securely benefit from HPC resources.

6. Conclusion

The EZ HPC architecture is designed with flexibility, extensibility, and security in mind. The current design is believed to fulfill these design goals. In addition, the architecture will be continuously improved to provide additional scalability and robustness.